Novel steganographic method based on jig swap puzzle images

En-Jung Farn
Chaur-Chin Chen
National Tsing Hua University
Department of Computer Science
Hsinchu, Taiwan 30013
E-mail: cchen@cs.nthu.edu.tw

Abstract. *A new steganographic method for data hiding in jig swap puzzle images is proposed. First, a color image is taken as input and divided into blocks. Second, each block is rearranged to a new position according to the secret data and a stegokey. The resulting image is a perfect jig swap puzzle. The original image is needed for extracting the secret data. Under the assumption that the receiver and the sender share some common images, the receiver can extract the secret data from the jig swap puzzle image. We also present a scenario for secret data transmission based on an online jig swap puzzle. Experimental results show that the proposed method is undetectable and robust under lossy image compression and format conversion.*
 [DOI: 10.1117/1.3073979]

1 Introduction

Steganography[1–4] is the art of concealing the existence of messages within seemingly innocuous carriers. A steganographic message is often embedded in a carrier, called a cover-carrier, resulting in a stego-carrier. Only the recipient can recover the message.

There are three important requirements in steganography. The first is imperceptibility: the difference between the stego-carrier and the cover-carrier should be imperceptible to a human perceptual system. The second is undetectability and security: the hidden message should not be detected by an inspector under visual and statistical attacks,[4–11] and it cannot be extracted without the stegokey. The third is robustness under limited modification, such as lossy image compression and format conversion.

In steganography, many kinds of carriers, such as text document, images, audio, and videos, can be used as covers. Among these carriers, the most popular one is images because it is relatively easy to remove redundant or noisy data from an image and replace with a hidden message. There are many steganographic methods[1–4] proposed for various kinds of images. Several steganographic tools[12–16] on the Internet are also available for varied image formats. Grayscale image domain, palette-based, and JPEG are the three most commonly used formats. The methods based on grayscale image domain[17–19] typically insert secret messages into the least significant bits (LSBs) of each pixel. LSB insertion is a quick and easy way to hide a message; unfortunately, it is vulnerable to lossy compression. For palette-based images, directly embedding messages in those indices will cause radical color change, since two neighboring colors in the palette may not look the same. Many efforts have been made to try to reduce the distortion created in the embedding process. S-Tools[12] reduces the number of colors to about 32 colors: message-bits are embedded in the LSB of each RGB channel, and thus, the color of each pixel does not change drastically. The drawback is that the palette is modified and a specific pattern will be found in a sorted palette. Without modifying the palette, Machado[20] proposed an approach to hide message in palette images by permuting the palette and then changing the LSBs in the palette indices. This approach works well only in grayscale images and images with related colors. Fridrich and Du[21] proposed a scheme based on the parity of palette colors. The methods based on JPEG images[14–16,22–25] typically hide the secret message in discrete cosine transform (DCT) coefficients. Upham's JSteg[14] sequentially replaces the LSB of DCT coefficients with the message's data. The χ^2-test[7] successfully detects the steganographic system. Instead of replacing the least significant bit of a DCT coefficient with message data, F5, proposed by Westfeld,[24] decrements its absolute value in a process. As a result, the χ^2-test cannot detect F5. However, Fridrich[9] and her group presented a steganalytic method that does detect images with F5 content.

As we know, the more kinds of images a steganographic system can use to embed data, the more secure the system is. In this paper, we propose a novel steganographic method that uses a special kind of images called jig swap puzzle images to embed secret messages. First, an image is divided into blocks. Second, each block is rearranged to a new position, and the neighboring relation between two blocks is broken. The secret message is embedded through the block rearranging process; the new position for each block is decided according to the secret message and a random number generator with a stegokey as its seed. This stegokey will be known by both sender and receiver. The resulting image is a perfect jig swap puzzle image and can be transmitted to the receiver directly. The original image is needed for extracting the secret message. To make the transmission more natural, we also present a scenario for secret message transmission. First, the sender creates an

Paper 08010RR received Jan. 25, 2008; revised manuscript received Nov. 29, 2008; accepted for publication Dec. 3, 2008; published online Jan. 29, 2009.

1017-9909/2009/18(1)/013003/10/$25.00 © 2009 SPIE and IS&T.

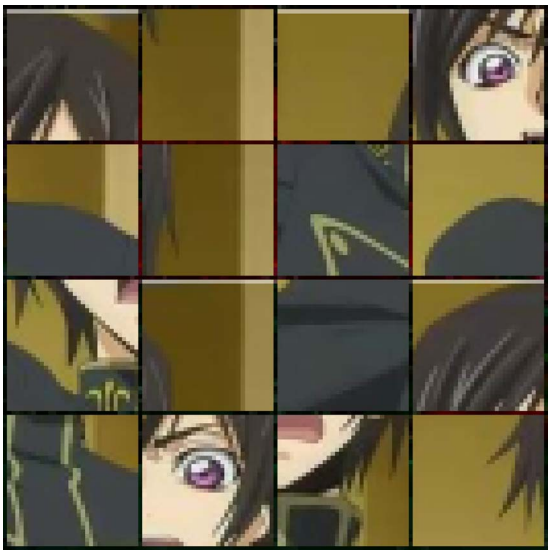


Fig. 1 A jig swap puzzle image.

online jig swap puzzle website, and then he or she will post the jig swap puzzle image with the secret message embedded on the website. The original image will also be put on the website. When the receiver is invited to play this jig swap puzzle game, he or she will open the webpage, this jig swap puzzle image and the original image can be downloaded, and the receiver can then extract the secret message. Experimental results show that the proposed method is undetectable and robust under image compression and format conversion.

The rest of this paper is organized as follows: Section 2 describes the proposed method and scenario. In Sec. 3, we analyze the robustness, undetectability, and security of the proposed method. Conclusions are given in Sec. 4.

2 Proposed Method and Scenario

A jig swap puzzle image (see Fig. 1) is formed by dividing a meaningful image into several blocks and randomly rearranging these blocks—it is a kind of puzzle games.[26] The original image is usually needed as a reference for a player;[26–32] otherwise, it will be difficult for the player, who has never seen the original image, to rearrange the jig swap puzzle image into the original one. In this paper, we propose a method to embed a secret message in a jig swap puzzle image. In the following, we will first describe the idea of secret message embedding and then give an example to demonstrate the embedding process.

2.1 *Idea of Embedding a Secret message in a Jig Swap Puzzle Image*

First, a 128×128 image [see Fig. 2(a)] is taken as input and divided into four 64×64 blocks. Then, each block is indexed from left to right and bottom to top [see Fig. 2(b)]. If we swap block 0 with block 2, we can get a jig swap puzzle image [see Fig. 2(c)]. Different block swapping will create a different jig swap image. To differentiate these jig swap puzzle images, each is assigned a permutation to uniquely represent the image. The permutation is created by concatenating the new position of each block from left

(a) (b) (c)

Fig. 2 A jig swap puzzle image and its corresponding permutation. (a) A digital image. (b) Four blocks in (a). (c) The result of applying permutation (2,1,0,3) to (b).

to right and bottom to top. In Fig. 2(c), since the new positions of blocks 0, 1, 2, 3 are 2, 1, 0, 3, respectively, the corresponding permutation of Fig. 2(c) is (2,1,0,3).

There are 24 (4!) kinds of permutations for the four blocks in this example. Each permutation will correspond to a jig swap puzzle image. These permutations are sorted in lexicographic order. The sorted order is used to index these permutations (see Table 1), and the index of a permutation is used to represent the secret data. Since $\log_2 24 \approx 4.584$, we take up to 4-bit secret data message and change it into a decimal number, which will then be used as the index to find the corresponding permutation. Since each permutation refers to a unique jig swap puzzle image, we can embed 4-bit secret data by creating the corresponding jig swap puzzle image. Figure 2(c) embeds 4-bit secret data 1110 (14).

Before describing the proposed method, we will first define a special kind of jig swap puzzle that will be referred to later.

Definition 1: Let A be an image, which is divided into several blocks. Let T be a permutation and B be a jig swap

Table 1 The lexicographic order for all permutations of four blocks.

Index	Permutation	Index	Permutation
0	(0,1,2,3)	12	(2,0,1,3)
1	(0,1,3,2)	13	(2,0,3,1)
2	(0,2,1,3)	14	(2,1,0,3)
3	(0,2,3,1)	15	(2,1,3,0)
4	(0,3,1,2)	16	(2,3,0,1)
5	(0,3,2,1)	17	(2,3,1,0)
6	(1,0,2,3)	18	(3,0,1,2)
7	(1,0,3,2)	19	(3,0,2,1)
8	(1,2,0,3)	20	(3,1,0,2)
9	(1,2,3,0)	21	(3,1,2,0)
10	(1,3,0,2)	22	(3,2,0,1)
11	(1,3,2,0)	23	(3,2,1,0)

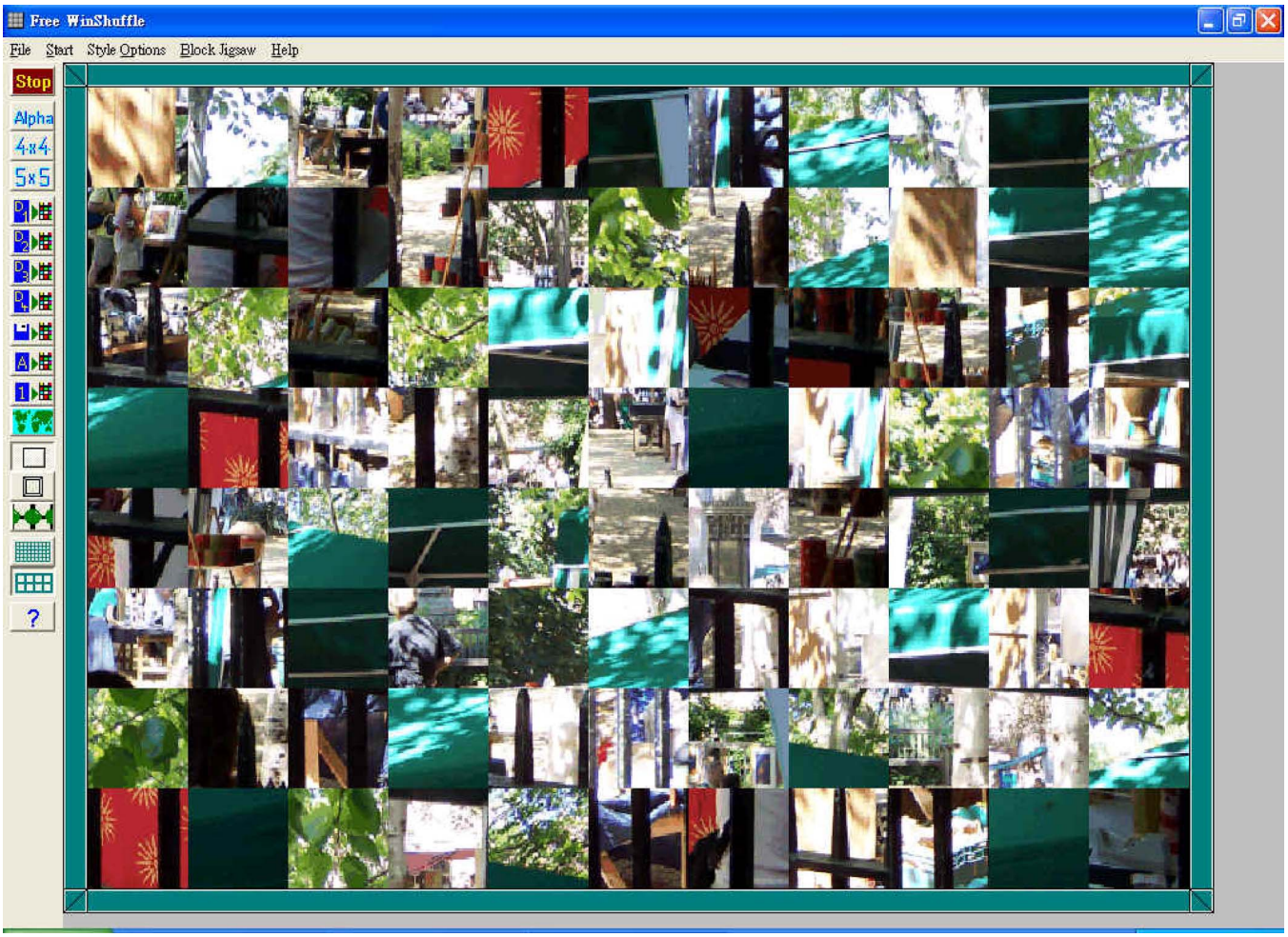


Fig. 3 A perfect jig swap puzzle from Ref. 28

puzzle obtained by applying T to blocks of A. Then, T is called a perfect permutation and B is called a perfect jig swap puzzle, if the following two conditions are satisfied: **C1:** The position of each block of B is not in the same position of A. **C2:** The sequence of the neighboring block in B is not the same as A.

From our observation, we find that a jig swap puzzle appearing in a jig swap puzzle game[27–29] is usually a perfect or near-perfect jig swap puzzle. That is, all or almost all blocks are not in their original positions, and neighboring block relations in the original image are also broken. Figure 3 shows a perfect jig swap puzzle from Ref. 28. Note that all jig swap puzzles created by our method will be perfect; this point will be explained in the next section.

2.2 *Proposed Embedding Process*

Based on the aforementioned idea, we propose a secret message embedding process. Figure 4 shows the block diagram of the proposed embedding process. Suppose that a secret message is represented in character form. Procedure 1 is provided to convert the secret message to a decimal number P according to the ASCII code of each character in the secret message.

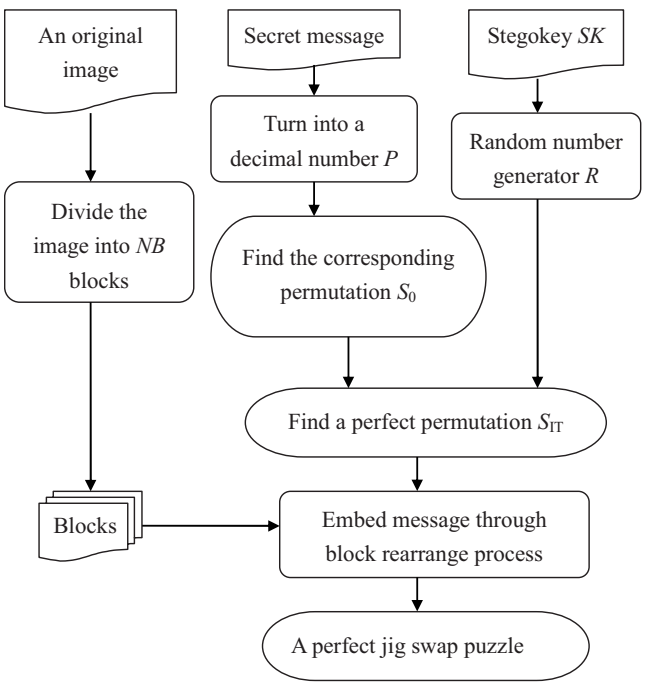


Fig. 4 The block diagram of the proposed embedding process.

Procedure 1 (Converting secret message to decimal number P)

$P=0$,
for $i=0$ to *secret message length*
$P=P$*256+ASCII code of i'th char in the secret message.

On the other hand, according to the message length, an $N\times N$ image with its size big enough to embed the message is chosen. The image is divided into blocks of $n\times n$. Let the number of blocks be NB, and then $NB=(N\times N)/(n\times n)$. Since one block will be chosen to store a specific number, which will be explained later, only $NB-1$ blocks can be used for permutation. Thus, the embedding capacity (EC) of the image is $\lfloor \log_2 (NB-1)! \rfloor$ bits. If $N=256$, $n=16$, $EC=1675$ bits. Let $M=\{(s_0,s_1,\ldots,s_i,\ldots,s_{NB-2})\,|\,0\leqslant s_i\leqslant NB-2\}$ be the set of all permutations of $NB-1$ numbers from 0 to $NB-2$.

After obtaining P, we can find the corresponding permutation, S_0, with index P. Let $S_0=(s_0^0,s_1^0,\ldots,s_{NB-2}^0)$. Since all permutations are sorted in lexicographic order, there exist at least $s_0^0\times(NB-2)!$ and at most $(s_0^0+1)\times(NB-2)!-1$ permutations before S. That is, $s_0^0\times(NB-2)!\leqslant P<(s_0^0+1)\times(NB-2)!$ This leads to

$$s_0^0=\left\lfloor \frac{P}{(NB-2)!} \right\rfloor. \tag{1}$$

In a similar way, we can get the remaining s_i^0. The details are described in Procedure 2.

Procedure 2 (Converting index P to permutation S)

```
for i=0 to NB-2
  N_level[i]=(NB-2-i)!
for i=0 to NB-2
  used[i]=false
for i=0 to NB-2
  temp=⌊P/N_level[i]⌋
  for j=0 to NB-2
    if (used[j]==false)
      if (temp==0)
        S_i^0=j
        used[j]=true
        break
      else
        temp--
P=P mod N_level[i]
```

If we simply use S_0 to generate a jig swap puzzle image, it may not be a perfect jig swap puzzle. In order to get a perfect jig swap puzzle, we will iteratively repermute S_0 until a perfect permutation is found. Here, we will use $JSPI_i$ to represent the jig swap puzzle image generated by S_0 through i times repermutation. The details are described in Procedure 3.

Note that in the i'th iteration, block bn in $JSPI_i$ stores block i (if $i\geqslant bn$, we store block $i+1$ instead to avoid block i staying in the original position) of the original image to indicate the number of iterations. And IT stores the number of iterations needed to get a perfect jig swap puzzle. In our experiments, we can always obtain a perfect jig swap puzzle image within 250 iterations; this point will be shown in Sec. 3.

Procedure 3 (Finding a perfect permutation)

Step 1: Let R be a random number generator, SK be a stegokey, and $i=0$.

Step 2: Use R with SK as its seed to generate a block number bn with $bn<NB$.

Step 3: Form a block sequence BS_i as

$$BS_i=\begin{cases}(0,1,\ldots,i-1,i+1,\ldots,NB-1) & \text{if } i<bn,\\ (0,1,\ldots,i,i+2,\ldots,NB-1) & \text{if } i\geqslant bn.\end{cases}$$

Step 4: Apply S_i to BS_i to get a new sequence

$$BS_i'=(b_0^i,b_1^i,\ldots,b_{NB-2}^i).$$

Step 5: Based on BS_i', a jig swap puzzle image $JSPI_i$ is created as follows:

For $i<bn$,

$$\begin{cases}\text{put block } b_j^i \text{ of the original image in block } j & \forall j<bn,\\ \text{put block } i \text{ of the original image in block } bn & \\ \text{put block } b_j^i \text{ of the original image in block } j+1 & \forall j<bn,\end{cases}$$

For $i\geqslant bn$,

$$\begin{cases}\text{put block } b_j^i \text{ of the original image in block } j & \forall j<bn,\\ \text{put block } i+1 \text{ of the original image in block } bn & \\ \text{put block } b_j^i \text{ of the original image in block } j+1 & \forall j<bn.\end{cases}$$

Step 6: Check if $JSPI_i$ is a perfect jig swap puzzle image. If yes, set $IT=i$, END.

Otherwise, $i=i+1$, use R to generate a permutation $T_i=(t_0^i,t_I^i,\ldots t_{NB-2}^i)$.

Step 7: Use Procedure 4 to combine T_i and S_{i-1} to get a new permutation S_i.

Go to step 3.

Procedure 4 (Combining two permutations)

input: $S=(s_0,s_1,\ldots,s_{NB-2})$, $T=(t_0,t_1,\ldots t_{NB-2})$
output: $S'=(s_0',s_1',\ldots,s_{NB-2}')$

```
for i=0 to NB-2
  temp_index=s_i
  s_i'=t_temp_index
```

2.3 *Example*

In the following, we will give an example to provide a more detailed explanation. Figure 5(a) shows a 128×128 original image. Suppose that the secret message is "test" and $n=32$, then $NB=(128\times128)/(32\times32)=16$ and the embedding capacity is $\lfloor \log_2 15! \rfloor=40$ bits. We first use Procedure 1 to convert the secret message "test" to the decimal number $P=1952805748$. Next, Procedure 2 is used to find the corresponding permutation $S_0=(0,1,6,2,14,4,8,11,9,5,7,3,13,10,12)$. Then Procedure 3 is conducted, and the random number generator R provided by the C++ standard library with seed $SK=179365$ is used to generate $bn(bn=1)$. Initially, $JSPI_0$ is an empty image. We place block 0 of the original image in block 1 of $JSPI_0$, and then S_0 is used to permute the remaining blocks of the original image, and the permuted

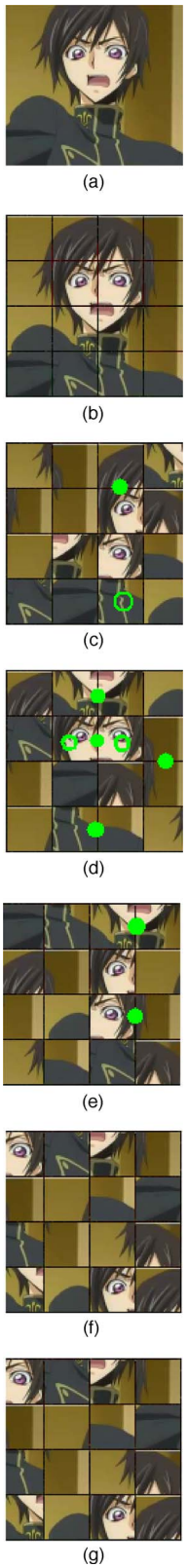


Fig. 5 An example to illustrate the secret message embedding process. (a) The original image. (b) The result of dividing (a) into blocks. (c) $JSPI_0$. (d) $JSPI_1$. (e) $JSPI_2$. (f) $JSPI_3$. (g) The compressed version of (f). (Color online only.)

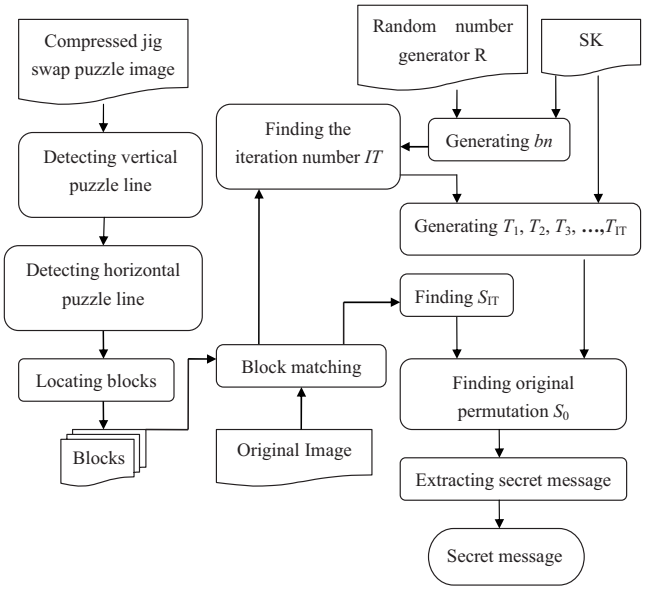


Fig. 6 The block diagram of the proposed extraction process.

blocks are put on $JSPI_0$ [see Fig. 5(c)]. Note that an open green circle in Fig. 5 indicates for a block staying in the original position and a solid green circle indicates two blocks keeping their neighboring relationship. (Color in online version only.) Since $JSPI_0$ is not a perfect jig swap puzzle, we use R to generate a permutation $T_1 = (4,7,14,0,13,9,1,6,3,8,5,2,10,11,12)$. Then, Procedure 4 is used to calculate the new permutation S_1. After S_1 is obtained, we place block 2 of Fig. 5(b) (block 1 is not used in order to avoid it staying in the original position) on block 1 of $JSPI_1$ and apply S_1 to permute the remaining blocks of Fig. 5(b) to get $JSPI_1$ [see Fig. 5(d)]. Since $JSPI_1$ is not a perfect jig swap puzzle, we use R to generate another permutation T_2 and obtain S_2. Then, block 3 of Fig. 5(b) is put on block 1 of $JSPI_2$, and S_2 is used to permute the remaining blocks of Fig. 5(b) to get $JSPI_2$ [see Fig. 5(e)]. Since $JSPI_2$ is not a perfect jig swap puzzle, we use R to generate T_3 and obtain S_3. Then, we place block 4 of Fig. 5(b) on block 1 of $JSPI_3$ and apply S_3 to permute the remaining blocks of Fig. 5(b) to get $JSPI_3$ [see Fig. 5(f)]. $JSPI_3$ is a perfect jig swap puzzle, and $IT=3$. In general, an image will be stored in compressed type. Thus, the created jig swap puzzle image will be compressed. Figure 5(g) shows the compressed version of Fig. 5(f). Notice that we have inserted a vertical/horizontal black line between two blocks.

2.4 *Extraction Process*

Here, we will provide a method to extract the secret message from the compressed jig swap puzzle image. Figure 6 shows the block diagram of the proposed extraction process. Two images are needed: one is the original image, and the other is the jig swap puzzle image. The extraction process contains two parts. The first part is to determine the block size and then to locate each block. This can be completed by locating black puzzle lines. The second part is to determine the corresponding block mapping between the

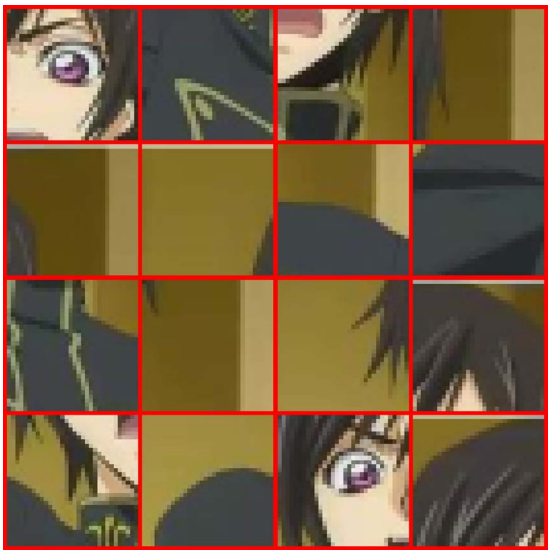


Fig. 7 The result of locating puzzle lines marked by red color. (Color online only.)

jig swap puzzle image and the original one, and this mapping is then used to obtain the secret message.

In the first part, for each row (column) in the jig swap puzzle image, we count the percentage of black pixels appearing in each row (column) as follows:

$$\text{For row } y, \quad BRP(y) = \frac{\sum_{x=0}^{N-1} Black(x,y)}{N}, \tag{2}$$

$$\text{For column } x, \quad BCP(x) = \frac{\sum_{y=0}^{N-1} Black(x,y)}{N}, \tag{3}$$

$$Black(x,y) = \begin{cases} 1 & Pixel(x,y) < 25 \\ 0 & Pixel(x,y) \geq 25, \end{cases} \tag{4}$$

where $Pixel(x,y)$ is the intensity of the pixel at (x,y), and N is the total number of pixels in a row or column.

If $BRP(y)$ [or $BCP(x)$] is over 60%, we consider that there is a horizontal (vertical) puzzle line at row y (column x). Figure 7 shows the detected puzzle lines in Fig. 5(g) drawn in red (color online only). Based on these located puzzle lines, we can obtain the block size and locate each block in the original and the jig swap puzzle images.

In the second part, we will find the corresponding block mapping between these two images to derive S_{IT}. Here, we will provide a block matching algorithm to find the mapping. Before describing the algorithm, we will first give some definitions:

Definition 2: Let $[R_i^s(x,y), G_i^s(x,y), B_i^s(x,y)]$ be the color of pixel (x,y) in block i of the jig swap puzzle image and $[R_j^o(x,y), G_j^o(x,y), B_j^o(x,y)]$ be the color of pixel (x,y) in block j of the original image. Each block has $n \times n$ size. Then, the difference between block i of the jig swap puzzle image and block j of the original image, $E(i,j)$, is defined as

$$E(i,j) = \sum_{x=0}^{n-1} \sum_{y=0}^{n-1} (|R_i^s(x,y) - R_j^o(x,y)| + |G_i^s(x,y) - G_j^o(x,y)| + |B_i^s(x,y) - B_j^o(x,y)|). \tag{5}$$

Definition 3: Let $H_k(i)$ be the block in the original image, which is the k'th most similar one to block i of the jig swap puzzle image. That is, there exist exactly $(k-1)$ blocks in the original image, with each block h having $E(i,h) \leqslant E[i,H_k(i)]$. Then, the difference between block $H_k(i)$ in the original image and block i in the jig swap puzzle image, $Dif(i,k)$, is defined to be

$$Dif(i,k) = E[i,H_k(i)]. \tag{6}$$

Definition 4: Define the block in the jig swap puzzle image that best matches block j of the original image to be $Match(j)$.

Definition 5: Let the difference between block j of the original image and its best matching block in the jig swap puzzle image be $Min(j)$, and then

$$Min(j) = E[Match(j), j]. \tag{7}$$

Definition 6: Define the number of blocks in the original image that are most similar to block i of the jig swap puzzle image and that have been matched to other blocks of the jig swap puzzle image during processing block matching to be $Used\_Block\_No(i)$.

Based on these definitions, we will provide a fast algorithm to find the best block matching. The idea is that if more than one block in the jig swap puzzle has the same best matched block, the block with the minimum difference wins, and others are rematched to the block with the next smallest difference according to the H array. We repeat this process until all blocks in the jig swap puzzle are matched. The details of this algorithm are described in Procedure 5.

Procedure 5 (Block matching algorithm)

```
for j=0 to NB=1 #j is the block index in the original image
  Match(j)=null
  Min(j)=∞
for i=0 to NB−1 #i is the block index in the jig swap puzzle image
  Used_Block_No(i)=0
for i=0 to NB−2
  p=i
  while{Min[H_{Used_Block_No(p)+1}(P)]!=∞}
    if{Min[H_{Used_Block_No(p)+1}(p)]
>Dif[p,H_{Used_Block_No(p)+1}(p)]}
      Min[H_{Used_Block_No(p)+1}(p)]
=Dif[p,H_{Used_Block_No(p)+1}(p)]
      temp=Match[H_{Used_Block_No(p)+1}(p)]
      Match[H_{Used_Block_No(p)+1}(p)]=p
      p=temp
    Used_Block_No(p)++
  Min[H_{Used_Block_No(p)+1}(p)]=Dif[p,H_{Used_Block_No(p)+1}(p)]
  Match[H_{Used_Block_No(p)+1}(p)]=p
```

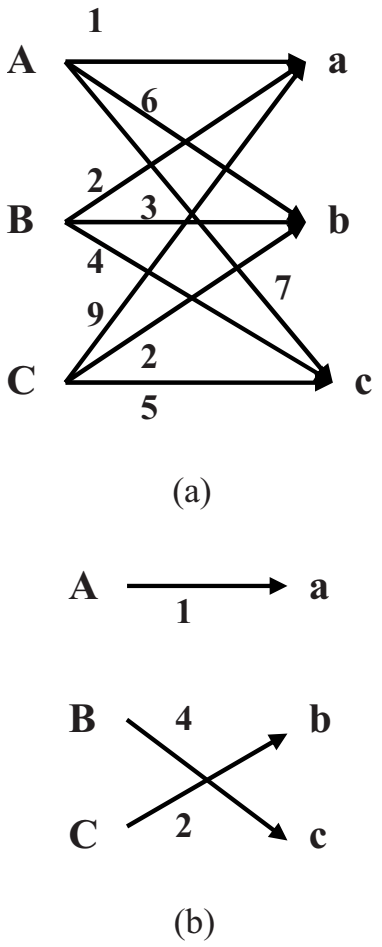


Fig. 8 A simple example to illustrate the matching idea. (a) The differences among A, B, C and a, b, c. (b) The final matching result.

A simple example is illustrated in Fig. 8. Figure 8(a) shows the differences between all pairs of blocks. First, A finds the best match a, $Match(a)=A$, $Min(a)=1$, and next, B finds the best match a. Since $E(B,a)=2>Min(a)$, B finds the second best match b and sets $Match(b)=B$, $Min(b)=3$. Then, C finds the best match b. Since $E(C,b)=2<Min(b)$, $Match(b)=C$, $Min(b)=2$, and b is rematched to C. B finds the third best matched c and sets $Match(c)=B$, $Min(c)=4$. Figure 8(b) shows the final matching result.

After applying Procedure 5, we can obtain the block matching $Match(j)$. Now, we use the random number generator and stegokey to generate bn, and then, we can obtain the iteration number IT as $IT=argi\ match(i)=bn$. And based on $Match(j)$, we can find S_{IT} with

$$s_j^{IT}=\begin{cases} Match(j) & \forall j<IT, \\ Match(j+1) & \forall IT\leqslant j<NB-1. \end{cases} \tag{8}$$

Using the generator R, we can obtain $T_1, T_2, \ldots, T_{IT}$. Based on $T_1, T_2, \ldots, T_{IT}$ and S_{IT}, we can find the original permutation S_0. The details are described in Procedure 6.

Procedure 6 (Finding the permutation S_0)
Generate the random number bn,
for $i=0$ to $NB-1$
  if$(Math(i)==bn)$
    $IT=i$ #iterative number
    break
for $j=0$ to $IT-1$
  $s_j^{IT}=Match(j)$
for $j=IT$ to $NB-2$
  $s_j^{IT}=Match(j+1)$
Generate permutations $T_1,T_2,\ldots,T_{IT}$
  use Procedure 4 to combine $T_1,T_2,\ldots,T_{IT}$ into a new permutation T'
for $j=0$ to $NB-2$
  for $i=0$ to $NB-2$
    if$(T_i'=S_j^{IT})$
      $s_j^0=i$
      break

Based on the obtained permutation S_0, we can calculate the index, P, of S_0. Let $S_0=(s_0^0,s_1^0,\cdots,s_{NB-2}^0)$, and then P can be evaluated as follows:

$$P=\sum_{i=0}^{NB-2}(s_i^0-|N_i|)\times(NB-2-i)!, \tag{9}$$

where

$$N_i=\{s_k^0|s_k^0<s_i^0,\ 0\leqslant k<i\},$$

$|N_i|$ = the number of elements in N_i.

Note that N_i consists of elements located before the i'th position in S_0 with their values less than s_i^0. For example, suppose that $S_0=(2,3,0,1)$, $N_0=\varnothing$, $N_1=\{2\}$, $N_2=\varnothing$, and $N_3=\{0\}$. After obtaining P, we can use Procedure 7 to convert P to the secret message.

Procedure 7 (Converting P to secret message)
  $i=0$
  while$(P>0)$
    secret_message$(i++)=P\%256$
    $P=\lfloor P/256\rfloor$
  reverse secret_message

2.5 *Proposed Scenario*

There are many puzzle game websites.[26–32] These websites either sell puzzle game programs[28–31] or provide online puzzle games.[26,32] Based on this fact and the proposed method described in Secs. 2.2–2.4, we will propose a scenario in which a secret message can be transmitted through an online jig swap puzzle website. Suppose that Alice wants to communicate with Bob. Under the proposed scenario, they should obey the following rules:

1. Alice and Bob both share a stegokey.
2. Alice will create an online jig swap puzzle website similar to Refs. 26 and 32, which provides users many kinds of jig swap puzzles to play online.
3. On the website, a user can select any jig swap puzzle to play.

Fig. 9 A simplified jig swap puzzle webpage.

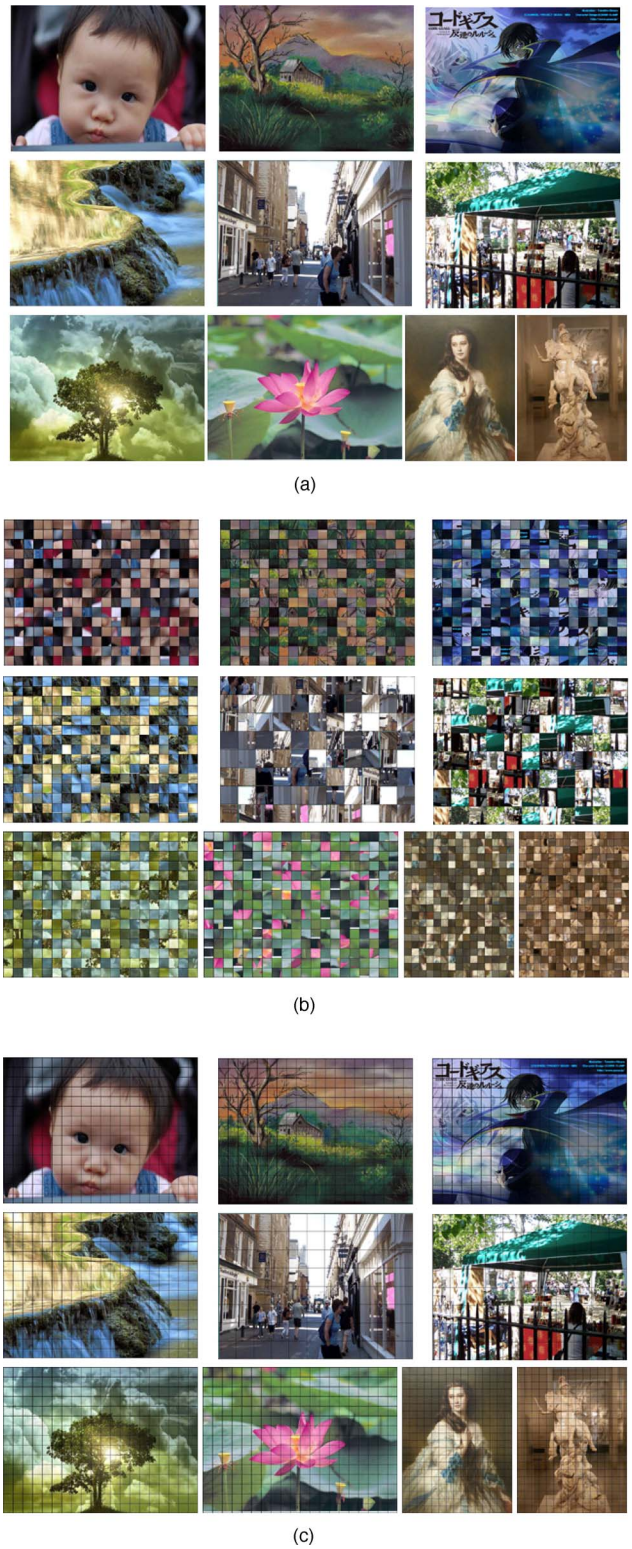
4. The jig swap puzzle webpage will be divided into two sections: left and right. The left section contains the original image of the selected jig swap puzzle, and the right section has the jig swap puzzle image with all blocks swappable. Figure 9 shows an example of a simplified webpage. When a user opens this webpage, the web browser has downloaded the original image and the jig swap puzzle image on the user's computer. At this time, the user can obtain these two images.
5. When Alice wants to send a secret message to Bob, she will select an image in which to embed the message using the proposed embedding process and the shared stegokey. The created jig swap puzzle image will then be put on the website as one of the online jig swap puzzle games.
6. Alice will then send an invitation e-mail to Bob. The e-mail will contain the web address in connection to the online jig swap puzzle with the secret message embedded.
7. When Bob receives Alice's e-mail, he will link to Alice's website mentioned in the invitation e-mail to get the compressed jig swap puzzle image and the original image.
8. Bob will then extract the embedded secret message using the proposed extraction process and the shared stegokey.

3 Analysis of the Proposed Method

The proposed method satisfies the three requirements of steganography: robustness, imperceptibility, and undetectability and security.

3.1 *Robustness*

The proposed method is robust under image compression and format conversion, including JPEG to GIF and GIF to JPEG. The reason is that the secret messages are embedded through block swapping and the block positions will not be changed by image compression and format conversion. To illustrate this point, for each created jig swap puzzle image, we use Adobe Photoshop CS to store the image in JPEG format with quality level High (7) and Medium (5) and in GIF format. We have taken 10 images, shown in Fig. 10(a), to conduct experiments. Each image is converted to 300



(a)

(b)

(c)

Fig. 10 The 10 test images used to show the robustness of the proposed method. (a) Original images. (b) The created perfect jig swap puzzle images. (c) The reconstructed images.

different perfect jig swap puzzles, one of which is shown in Fig. 10(b). Each created jig swap puzzle image is stored in JPEG format with quality level (7 and 5) and in GIF format, and each JPEG image is converted to GIF format and

each GIF image is converted to JPEG format with quality level 5. Thus, each jig swap puzzle image is converted to six different compression versions, and we have a total of 18,000 jig swap puzzle images. By applying the proposed block matching algorithm to all these compressed jig swap puzzles, we can correctly obtain their corresponding permutations. Some reconstructed images are shown in Fig. 10(c). Experimental results show that the proposed method is robust to the lossy image compression and format conversion.

3.2 *Imperceptibility*

Each jig swap puzzle image created by the proposed method is a perfect jig swap puzzle image like those appearing in the jig swap puzzle games; thus, our method satisfies the imperceptibility requirement.

3.3 *Undetectability and Security*

Security refers to the inability of an eavesdropper to detect hidden information. In practice, a steganographic scheme is considered secure if no existing attack can be modified to build a detector that would be able to distinguish between cover and stego images with a success better than random guessing.[4] Let the probability distribution of cover images be P_c and the probability distribution of stego-images be P_s. According to Cachin's definition[5] for a secure stegosystem, if $P_c=P_s$, then the stegosystem is perfectly secure, because it is impossible for a warden to distinguish between the cover-image and the stego-image. Note that we embed secret data via a jig swap puzzle, and if the probability distribution of jig swap puzzle images appearing on websites and the probability distribution of the jig swap puzzle images created by the proposed method are identical, then the proposed method can be considered secure.

Here, we will show that the proposed method is secure and undetectable. Until now, although there are many jig swap puzzle games on the Internet, there are no rules restricting swapping methods. That is, no reference can be used to determine whether a jig swap puzzle image is abnormal. However, from our observation, most jig swap puzzles appearing in puzzle games are perfect or near perfect. Thus, we suppose that a natural jig swap puzzle image should be a perfect one and that the probability distribution of the natural perfect jig swap puzzle images is uniform—this means that each perfect permutation will have the same probability. We have described an iteration permutation method in Sec. 2.2 to make the created jig swap puzzle perfect. To illustrate this point, we have conducted many experiments. For each kind of block number $k\times k$ with $k=16,32$, we randomly generate 3,000,000 numbers from $[0,(k\times k-1)!-1]$ as secret data. Next, each number is considered as a permutation index and converted to the corresponding permutation S_0. Then random permutation is applied iteratively until a perfect jig swap puzzle is obtained. The numbers of iterations are shown in Table 2. From this table, we can see that for block number 16×16, 2,819,721 secret data numbers can obtain perfect jig swap puzzles within 50 iterations. And for each secret data number, we can always get a perfect jig swap puzzle within at most 250 iterations. As described in Sec. 2.2, we store the iteration number IT by putting block IT of the original image in block bn of the created jig swap puzzle; thus, IT should be less than k^2. Since $250<k^2$ for $k\geq 16$, this evidence demonstrates that the proposed method is visually and statistically undetectable. Furthermore, our experimental results also show that the 3,000,000 created perfect jig swap puzzle images are all different. This means that the probability distribution of the created perfect jig swap puzzle images is uniform and matches the probability distribution of the natural perfect jig swap puzzle images.

Table 2 The iteration numbers for finding perfect jig swap puzzles.

Block number	16×16	32×32
Iteration number needed	Number of secret data	
0–50	2819721	2776739
51–100	162822	192582
101–150	15474	20304
151–200	1868	10330
201–250	115	45

Existing attacks[6–11] try to find some statistical properties of gray values in an image to determine whether the image hides a secret message. However, in the proposed method, for a created jig swap puzzle image, except for blocks swapped, the color values of pixels are exactly the same as those in the original image. Thus, the proposed method is immune from those existing attacks. Furthermore, since the proposed method uses a stegokey as the seed of a random number generator to adjust the original permutation, even if an attacker has enough knowledge of the proposed method and can guess the iteration number, without the stegokey, the attacker still cannot get the original permutation.

Based on the previously described reasons, we can conclude that the proposed method is undetectable and secure.

4 Conclusions

In this paper, we have presented a novel steganographic method based on jig swap puzzle images. A secret message is hidden by creating a perfect jig swap puzzle image. Experimental results show that the proposed method is robust to image compression and format conversion. The proposed method also satisfies the other two requirements: imperceptibility, and undetectability and security. Based on the proposed method, a scenario has been provided. Under this scenario, the secret message can be transmitted through an online jig swap puzzle website. In this way, an innocent user will think that the website provides only online jig swap puzzle games. Furthermore, because there are many jig swap puzzle images on the website, guessing which one embeds a secret message is difficult. Note that if we do not want to establish a website, the proposed method still works by directly sending the created jig swap puzzle image to the communicative party under the assumption that the receiver and the sender share some common images.

On the other hand, to reconstruct a jig swap puzzle without the original image is an nonpolynomial (NP) complete

problem.[33] Several references[34–36] provide methods to solve jig swap puzzles, but no one can solve this problem completely. If someone wants to use our proposed method to hide data and will not provide the original image for data extraction, he or she can use the method proposed by Nielsen *et al.*[34] to solve the created jig swap puzzle partially and can then manually do some adjustment to get the correct permutation.

Acknowledgments

This work is supported by NSC Grant No. 97-2221-E-007-122-MY3. The authors would like to thank the anonymous reviewers for their many valuable suggestions, which have greatly improved the presentation of the paper.

References

1. W. Bender, D. Gruhl, N. Morimoto, and A. Lu, "Techniques for data hiding," *IBM Syst. J.* **35**(3), 313–336 (1996).
2. N. Johnson and S. Jajodia, "Exploring steganography: seeing the unseen," *IEEE Comput. Sci. Eng.* **31**(2), 26–34 (1998).
3. N. Provos and P. Honeyman, "Hide and seek: an introduction to steganography," *IEEE Security Privacy* **1**, 32–44 (2003).
4. I. J. Cox, M. L. Miller, J. A. Bloom, J. Fridrich, and T. Kalker, *Digital Watermarking and Steganography*, pp. 429–495, Morgan Kaufmann, San Francisco (2008).
5. C. Cachin, "An information-theoretic model for steganography," in *2nd Int. Workshop on Information Hiding*, Portland, Oregon, Lecture Notes in Computer Science, vol. 1525, pp. 306–318, Springer-Verlag, Berlin (1998).
6. N. F. Johnson and S. Jajodia, "Steganalysis of images created using current steganographic software," in *Proc. 2nd Int. Workshop on Information Hiding*, Portland, Oregon, Lecture Notes in Computer Science, vol. 1525, pp. 273–289, Springer-Verlag, Berlin (1998).
7. A. Westfeld and A. Pfitzmann, "Attacks on steganographic systems," in *Proc. 3rd Int. Workshop on Information Hiding*, Dresden, Germany, Lecture Notes in Computer Science, vol. 1768, , pp. 61–76, Springer-Verlag, Berlin (1999).
8. S. Lyu and H. Farid, "Detecting hidden messages using higher-order statistics and support vector machines," in *Proc. 5th Int. Workshop on Information Hiding*, Noordwijkerhout, the Netherlands, Lecture Notes in Computer Science, vol. 2578, pp. 340–354, Springer-Verlag, Berlin (2002).
9. J. Fridrich, M. Goljan, and D. Hogea, "Steganalysis of JPEG images: breaking the F5 algorithm," in *Proc. 5th Int. Workshop on Information Hiding*, Noordwijkerhout, the Netherlands, Lecture Notes in Computer Science, vol. 2578, pp. 310–323, Springer-Verlag, Berlin (2002).
10. S. Dumitrescu, X. Wu, and Z. Wang, "Detection of LSB steganography via sample pair analysis," in *Proc. 5th Int. Workshop on Information Hiding*, Noordwijkerhout, the Netherlands, Lecture Notes in Computer Science, vol. 2578, pp. 355–372, Springer-Verlag, Berlin (2002).
11. J. Fridrich, T. S. Holotyak, and D. Soukal, "Stochastic approach to secret message length estimation in ±k embedding steganography," *Proc. SPIE* **16**(20), 673–684 (2005).
12. A. Brown, "S-Tool V4," http://www.spychecker.com/program/stiils.html, Jan. 21 (2009).
13. H. Repp, "Hide4PGP," http://www.heinz-repp.onlinehome.de/hide4PGP.htm, Jan. 21 (2009).
14. D. Upham, "JPEG-JSteg," http://www.funet.fi/pub/crypt/stefanography/jpeg-jsteg-v4.diff.gzee (1997).
15. A. Latham, "Steganography: JPHIDE and JPSEEK," http://linux01.gwdg.de/~alatham/stego.html (1999).
16. N. Provos, "Outguess," http://www.outguesst.orgoutguest-0.2.tar.gz (2001).
17. Y. K. Lee and L. H. Chen, "High capacity image steganographic model," *IEE Proc. Vision Image Signal Process.* **147**(3), 288–294 (2000).
18. C. C. Thien and J. C. Lin, "A simple and high-hiding capacity method for hiding digit-by-digit data in images based on modulus function," *Pattern Recogn.*, **36**(12), 2875–2881 (2003).
19. C. K. Chan and L. M. Cheng, "Hiding data in images by simple LSB substitution," *Pattern Recogn.* **37**(3), 469–474 (2004).
20. R. Machado, "\Stego," http://www.stego.com, Jan. 21 (2009).
21. J. Fridrich and R. Du, "Secure steganographic methods for palette images," in *Proc. 3rd Information Hiding Workshop*, Dresden, Germany, Lecture Notes in Computer Science, Vol. 1768, pp. 47–60, Springer-Verlag, New York (2000).
22. L. M. Marvel Jr., C. G. Boncelet, and C. T. Retter, "Spread spectrum image steganography," *IEEE Trans. Image Process.*, **8**, 1075–1083 (1999).
23. N. Provos, "Defending against statistical steganalysis," in *Proc. 10th Usenix Security Symp.*, Washington, DC, pp. 323–335, Usenix Assoc. (2001).
24. A. Westfeld, "F5—a steganographic algorithm: high capacity despite better steganalysis," in *Proc. 4th Int. Workshop on Information Hiding*, Pittsburg, PA, Lecture Notes in Computer Science, vol. 2137, pp. 289–302, Springer-Verlag, Berlin (2001).
25. Y. K. Lee and L. H. Chen, "Secure error-free steganography for JPEG images," *Int. J. Pattern Recognit. Artif. Intell.*, **17**(6), 967–981 (2003).
26. http://www.jigzone.com, Jan. 21 (2009).
27. http://www.jigsawpuzzles.ws, Jan. 21 (2009).
28. http://www.pajersoft.com/winshweb.htm, Jan. 21 (2009).
29. http://www.kraisoft.com/puzzle-games/jigsaw, Jan. 21 (2009).
30. http://www.spintop-games.com/jigsaw_game_download/jigsaw_landscapes.html, Jan. 21 (2009).
31. http://www.spintop-games.com/jigsaw_game_download/jigsaw_nature.html, Jan. 21 (2009).
32. Yell, "Picture puzzle," http://www.onlinegameshq.com/game/867/picture-puzzle.html, Jan. 21 (2009).
33. E. D. Demaine and M. L. Demaine, "Jigsaw puzzles, edge matching, and polyomino packing: connections and complexity," *Graphs and Combinatorics* 23, 195–208 (2007).
34. T. R. Nielsen, P. Drewsen, and K. Hansen, "Solving jigsaw puzzles using image features," *Pattern Recogn. Lett.*, **14**(29), 1923–1933 (2008).
35. Y. X. Zhao, M. C. Su, Z. L. Chou, and J. Lee, "A puzzle solver and its application in speech descrambling," in *Proc. 2007 WSEAS Int. Conf. Computer Engineering and Applications*, Gold Coast, Australia, pp. 171–176, World Scientific and Engineering Academy and Society (WSEAS), Stevens Point, WI (2007).
36. A. Pal, K. Shanmugasundaram, and N. Memon, "Automated reassembly of fragmented images," *IEEE Trans. Image Process.*, **15**(2), 385–393 (2003).

En-Jung Farn received a BS degree from the Department of Computer Science and Information Engineering at Chiao Tung University, Hsinchu, Taiwan, in 2004. He is currently a PhD student in the Department of Computer Science, National Tsing Hua University. His current research interests include image processing and information security.

Chaur-Chin Chen received a BS degree in mathematics from National Taiwan University, Taipei, in 1977, and MS degrees in both mathematics and computer science and a PhD degree in computer science, all from Michigan State University (MSU), East Lansing, in 1982, 1984, and 1988, respectively. He worked as a research associate at MSU in 1989. He is currently a professor in the department of Computer Science at National Tsing Hua University. He was a visiting scholar at MSU in 1997. His current research interests are information hiding, biometrics, and microarray image data analysis.